Microcontrollers
"Assembly Comands"
"Directives"

Assembly is written as:

[label:] Mnemonic [Operands] [; comment]

→ between [ ] can be existed or can not.

Mnemonic	Comment	Memory	Mnemonic	Comment	Memory
ORG	ORG 500H ↓ address . indicates the beginning of the address "1st instruct. in the program" also can be used with DB directive	0		Ex: ORG 300H SQR_TABLE: DB 0, 1, 4, 9	
DB	DATA1: DB 28 DATA2: DB 39H DATA3: DB 00110101B DATA4: DB "2591" ↳ ASCII 1 Byte = character لكل - تخزين - بداية التخزين من العنوان اللي عليه الlabel . used to define 8-bit data	Specified by the data size ↓ RAM but for ROM the DB has 0	END	END . Indicates ~~the beg~~ the assembler that this is the source file end . It is the last line in source file . Anything after it is ignored	0

Mnemonic	Comment	Memory	Mnemonic	Comment	Memory
EQU	COUNT EQU 25 MOV R3, #COUNT أي حاجة اسمها COUNT بتتبدل بـ 25 (تتبدل text بـ text) • Used to define a constant without occupying a memory location.	0			
BIT	LED BIT P1.7 . It assigns P1.7 to be called LED within the code . It is used with single bits. . SW BIT P2.3 SW ↓ name, P2.3 ↓ the bit	0			

Microntrollers

"Assembly Commonds"
"Instructions"

Assembly is written as:

[label:] Mnemonic [Operands] [;comment]

=> between [ ] can

Mnemonic	Comment	Memory	Mnemonic	Comment	Memory
MOV	MOV dest, src • copies src into dest. • #23 → value • #23H → Hex. value • #0F9H ↳ to make the assembler understand that F is not a character	2 Bytes		• MOV A, #7F2H ↳ illegal because, #7F2H needs bigger than 8 bits • Affects CY "flag"	
	• MOV A, #5 ↳↳ A = 5H (= 5) = 05H = 0000 0101 ↳ it assumes the rest of a register is zeros as 5 only need 4 bits (0101) to be represented. • MOV R4, R7 → illegal between Rn registers		ADD	ADD A, src • A = A + src → it must be A and A only • Affects CY, OV, AC, P "Flags" • src could be register, address or immediate data • Memory to memory arith. operations are not allowed	1 Byte

Mnemonic	Comment	memory
PUSH	PUSH Address PUSH 6 ⇓ means memory location number 06H (6) "which is R6 also" • It increments SP by 1, ↓ stack Pointer then, puts the value within the address into the next place in the stack (which is pointed by SP after it's increment) • SP is 07 by default at the operation • ex: MOV R6, #25H PUSH 6 ⇓ SP = SP+1 = 7+1 = 8 (0A, 09, 08: 25H) from R6 which has the address 6 (06H) • PUSH A → invalid → only addresses	2 Bytes
POP	POP Address POP 7	2 Bytes

Mnemonic	Comment	memory
	• transfers the value of the current stack place pointed by SP to the address specified and decreases SP by one	
DJNZ	DJNZ reg., label decrement, Jump if not zero • decrements the register by 1 then it goes to the label if the register ≠ 0 ex: MOV R3, #10 NEXT: MOV R2, #70 AGAIN: CPL A DJNZ R2, AGAIN DJNZ R3, NEXT ⇓ 700 times loop • Label address must be within: -128 + PC + 127	Short Jump ↓ 2 Bytes

Mnemonic	Comment	Memory
JZ	JZ label Jump if zero • Jumps to label when A=0	Short Jump 2 Bytes
JNC	JNC label Jump if no carry • Jumps to label when CY=0	Short Jump 2 Bytes
CJNE	CJNE A, byte, label Conditional Jump if not equal • Jump if A ≠ Byte CJNE reg, #data, HERE	Short Jump 2 Bytes
JC	JC Label Jump if carry • Jump if CY=1	Short Jump 2 Bytes
JB	JB bit, Label • jump if this bit = 1	Short Jump 2 Bytes
JNB	JNB bit, label • jump if this bit ≠ 1	Short Jump 2 Bytes
SJMP	SJMP Label Short Jump • first byte is opcode • Second byte is relative 8-bit target address (-128 + PC + 127) in this range • This is unconditional jump instruction	2 Bytes
LJMP	LJMP Lable Long Jump • First byte is opcode • Second and third bytes are 16-bit target address // to any 11 place $-(2^{15}) + PC + (2^{15}-1)$	3 bytes

Mnemonic	Comment	Memory
ACALL	ACALL Label (Absolute Call) • 11 bits are used for address within 2k-byte range • when a subroutine is called, processor saves on the stack the address of the instruction immediately below the call instruction	2 Bytes
LCALL	LCALL Label (Long Call) • first byte is the opcode • Secord and third bytes are used for address of target subroutine which is located anywhere within 64k byte address space	3 Bytes

Mnemonic	Comment	Memory
RET	RET • Every subroutine needs RET as the last instruction to transfers control back to the caller • pops the return address from the stack into the PC and resumes within the instructions after the call instruction ex: LCALL AGAIN ORG 300H AGAIN: DJNZ R5, AGAIN RET ex: → page 116 ,117	1 Byte
NOP	NOP (No opcode) • Does nothing • No registers or flags are affected • used widely for delays • 1 machine cycle	1 Byte

Mnemonic	Comment	Memory
CPL	CPL bit ↳ complement • ex: CPL P1,2 • P1.2 = (P1.2)'	1 Bytes
SETB	SETB bit ↳ Set bit • ex; SETB P1.2 • P1.2 = 1	1 Bytes
CLR	CLR bit ↳ clear • ex; CLR P1.2 • P1.2 = 0	1 Byte
JBC	JBC bit, Label ↳ Jump if bit=1 and clear • Jumps to label if bit=1 then clears the bit after that,	2 Bytes "Short" Jump
ANL	ANL Δ, □ ↳ AND Latch • It reads internal latch of Δ "port" then brings data to CPU	2 Bytes
	• The data is Anded with □ and the result is returned back to the latch in order to affect the external pin ex: • ANL P1,A → P1 = P1 AND A • ANL P1, #55H * Read-Modify-Write	
ORL	ORL Δ, □ ↳ OR Latch Δ = Δ OR □ ↳ Port * Read-Modify-Write	2 Bytes
XRL	XRL Δ, □ ↳ XOR Latch Δ = Δ XOR □ ↳ port *Read-Modify-Write	2 Bytes

Mnemonic	Comment	Memory
MOVC	MOVC A, @A+DPTR ↓ Move code • used to load data from the ROM "look-up table" • A = Data which was in ROM at the location @A+DPTR ↳ points to some location	1 Byte
MOVX	MOVX @R0, A • used to transfer data between A and locations in external memory "when used"	1 Byte
INC	INC A • Increments a register or a memory location by 1	1 Byte

Mnemonic	Comment	Memory
DEC	DEC @R1 • Decrements a register or a memory location by 1	1 Byte
ADDC	ADDC A, #3H ↓ Add with Carry • A = A + 3H + C (C) ↓ Carry	1 Bytes
DA	DA A ↓ Decimal Adjust this is A only • It corrects the problem of BCD addition (adds 6 to A) to adjust 2 BCD numbers addition. • Works only after ADD instruction but not after INC • ex: ADD A, B DA A	1 Bytes

Mnemonic	Comment	memory
	• After ADD or ADDC instruction ① if the lower nibble is greater than 9 or [if AC=1], it adds 0110 (6) to the lower nibble ② if the upper nibble > 9 or [if CY=1], it adds 0110 (6) to the upper nibble	
SUBB	SUBB A, src subtract with borrow • A = A − SRC − CY • There is no SUB • to make only SUB operation, we have to make CY=0 prior to instruction execution • It tales 2's complement of src then adds it to A then complements CY	1 Byte

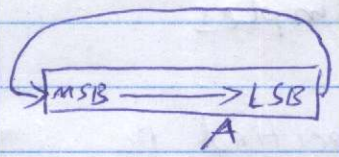
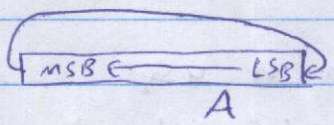
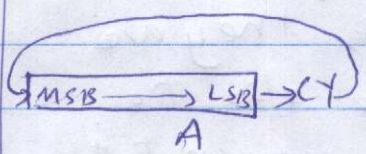
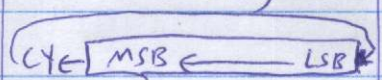
Mnemonic	Comment	memory

MUL	MUL AB
	• BA = A*B
	• It preforms byte by byte multiplication operation and it will result in a 16-bit result.
	• The result will be in the form:
	B = hig byte of result
	A = Low byte of result
	• Bytes are assumed to be unsigned
	• ex: (A,B)
	MOV A, #25H
	MOV B, #65H
	MUL AB
		1 Bytes

DIV	DIV AB • A = A/B (B = A%B ← remineder • if B=0 (division by zero) → OV=1 → indicating error	1 Byte

Mnemonic	Comment	Memory
	• Byte by Byte division • Bytes are assumed to be unsigned (A, B)	
ANL (2)	ANL dest, src (AND Logic) • performs Logic AND on src, dest, • ~~src~~ dest, is normally A • src can be register, address or imediate • can be used with bits	2 Bytes
ORL (2)	ORL dest, src (OR Logic) • dest. is normally A. • src can be register, address or immediate • can be used with bits	2 Bytes

Mnemonic	Comment	Memory
XRL (2)	XRL dest, src (XOR Logic) A register address immediate • can be used with bits	2 Bytes
CPL (2)	CPL A (complement) • complements A	1 Byte
CJNE (2)	~~CJNE src, dest~~ CJNE dest., src, Label A or Rn registers; register address immediate • the src, dest. remain the same and do not change • It changes CY flag dest. >, src → CY=0 dest < src → CY=1 • It is a subtraction operation in reality except the operands are unchanged	Short Jump 2 Bytes

Mnemonic	Comment	Memory
RR	RR A ↑ Rotate Right 	1 Byte
RL	RL A ↑ Rotate Left 	1 Bytes
RRC	RRC A ↑ Rotate Right with carry 	1 Byte
RLC	RLC A ↑ Rotate Left with carry 	1 Byte
Mov (2)	Mov P2.1, C ↓ Can be used with bits also	2 Bytes

Mnemonic	Comment	Memory
SWAP	SWAP A ↓ must be A . swaps lower nibble of A with the higher nibble of A	1 Byte
RETI	RETI . The last instruction at an Interrupt Service Routine (ISR) which returns back to the main program . It clears the interrupt-in-service flag also, it clears TCON.1, TCON.3 and it also indicates that the interrupt pin can accept another interupt, RET can not do that.	1 Bytes

Notes:

- Instructions that operates on single bits is called "single-bit instructions"
- Look at the examples
- Memory field specified is about the size that an instruction has within the ROM
- An Instruction which use 2 Bytes from ROM is called "2-Byte instruction" and so on.
- Directives do not consume any size of ROM, they are for the assembler also to make understand some notations or notes in the code.